Standard e Vincoli Architetturali

Controllo Emissione

	Ente Aziendale	Nome
Proposto da	DIDT/CTO/IEA	R. MARCHIANI
Verificato da	DIDT/CTO	F. FIASCHI
Autorizzato da	DIDT/CTO	F. FIASCHI

Registro delle modifiche

Rev.	Motivo	Autore delle modifiche		Data
		Ente Aziendale	Nome	
1.0	Prima stesura	ITS/SNF/SVS	F. FIASCHI	02/12/2013
2.0	Revisione in ottica ISO 27001	ITS/SNF/SVS	M. CALERI	18/04/2014
2.1	Revisione formato e aggiornamento dati	ITS/SNF/SVS	M. CALERI	23/06/2015
2.2	Aggiornamento versioni sw base	ITS/SNF/IIT/SVS	R. MARCHIANI	29/04/2016
2.3	Aggiunto paragrafo su browser compatibili	ITS/SNF/IIT/SVS	R.MARCHIANI/R.SQUARCI	8/11/2016
2.4	Aggiornamento sigle aziendali, versioni sw, modifiche minori di impaginazione	ITS/SNF/ITR/SES	R. MARCHIANI	26/4/2017

2.5	Aggiornamento capitoli 13 e 14	ITS/SNF/ITR/SES; ITS/SNF/SCR	R. MARCHIANI; M. CALERI	11/05/2017
2.6	Aggiornamento Capitolo 3.1 Capitolo 4 Capitolo 6.3	ITS/SNF/ITR/SES	S. CAVALIERI	21/05/2018
2.7	Aggiornamento Capitolo 4 Capitolo 11.2	ITS/SNF/ITR/SES	L. CARLÀ	30/05/2018
2.8	Aggiornamento Capitolo 2.6 Capitolo 6.3 Capitolo 7 Capitolo 10.3 Capitolo 11.1	ITS/SNF/ITR/SES	M. MARFELLA; S. CAVALIERI	31/05/2018
2.9	Aggiornamento: Capitolo 6.3.1 Capitolo 6.3.4 Inserimento: Capitolo 6.3.5 Capitolo 6.3.6	ITS/ITR/SES	S. CAVALIERI	03/05/2019
3.0	Aggiornamento: Capitolo 6.2.2 Capitolo 6.2.3 Capitolo 6.2.4	ITS/ITR/SES	L. CARLA'	22/05/2019
4.0	Riorganizzazione generale della struttura del documento con inserimento principi di	DIDT/CTO/IEA	R.MARCHIANI	29/11/2021

	standard architetturali			
4.1	Aggiornamento documento Creata nuova sezione linee guida di sviluppo Creata nuova sezione AI Architecture	DIDT/CTO/IEA	R.MARCHIANI	10/07/2024

Riferimenti

Rif.	Nome del documento	Descrizione
1	UNI - ISO	UNI CEI ISO/IEC 27001:2014 - Tecnologie informatiche - Tecniche per la sicurezza - Sistemi di gestione per la sicurezza delle informazioni – Requisiti
2	UNI - ISO	UNI CEI ISO/IEC 27002:2014 - Tecnologie informatiche - Tecniche per la sicurezza - Raccolta di prassi sui controlli per la sicurezza delle informazioni
3	ASPI_MN_ITS_SGSI	Manuale del Sistema di Gestione della Sicurezza delle Informazioni
4	AWS Well-Architected Framework	Documento guida sulle architetture realizzate sul cloud provider Amazon Web Services (AWS)
5	Procedura Gestionale di Gruppo – Modello per l'adozione e gestione delle soluzioni IT di Gruppo	Documento interno ASPI per l'individuazione delle modalità di collaborazione tra ASPI e le Società del Gruppo nell'ambito dello sviluppo e della gestione delle soluzioni e servizi IT

Indice

1 Scopo del documento

Lo scopo del seguente documento è quello di indicare gli standard architetturali IT da utilizzare in ASPI per lo sviluppo di applicazioni e servizi. Nel documento vengono anche segnalate *best practices* consigliate nel disegno di soluzioni IT.
Al paragrafo 7 del documento sono riportati i software di base supportati o in prossima dismissione (decomissioning) e le relative versioni.

2 Contesto

Il documento esplicita standard e buone regole di sviluppo architetturale in coerenza con il piano di trasformazione digitale "next to digital" (in seguito abbreviato come "NtD") 2020-2023 che ha definito le seguenti linee guida:

- **DevSecOps By Design**: adottare logiche di SecDevOps come abilitatori di sviluppo e deployment agile e sicuro di nuovi sistemi digitali attraverso l'implementazione di strumenti automatizzati (ad esempio, Build management, Test automation, Automated vulnerability scanning e strumenti di continuous delivery);

- **Platform First**: adottare il più possibile piattaforme rispetto a soluzioni *standalone*, al fine di standardizzare, consolidare, modernizzare e ridurre il panorama complessivo delle applicazioni, consentendo processi *end-to-end* (ad esempio nell'ambito Human Resources quello di "Hire-2-Retire"), al contrario di verticali specifici, fornendo un'esperienza utente unica lungo l'intero processo e nel frattempo attraverso decomissioning dei sistemi legacy esistenti;

- **Cloud First**: seguire un approccio per la migrazione al cloud che utilizzi una strategia cloud first per tutte le nuove applicazioni (sia package che custom cloud ready solutions o SaaS) e che preveda, per quelle esistenti (principalmente se influenzate dalla roadmap NtD) di effettuare un approccio di trasformazione che miri a beneficiare dei vantaggi competitivi e tecnici del cloud;

- **Data-centric**: Rendere disponibili i dati di ASPI attraverso la democratizzazione del Data Layer e la creazione di golden source di master-data con una forte capacità di governance, completando il data layer con le caratteristiche attualmente mancanti per la distribuzione e l'integrazione;

- **API First**: Implementare un approccio API first disaccoppiando i diversi livelli e sistemi attraverso strumenti di integrazione (ad esempio, gateway API), consentendo una standardizzazione e democratizzazione nell'accesso ai dati, così come la scalabilità e la riusabilità dei moduli architetturali evitando le interfacce punto-punto. L'obiettivo è di rendere l'accesso ai dati agnostico dalle applicazioni attraverso l'esposizione di end-point standardizzati;

- **Microservice Oriented**: Adottare un enterprise architecture orientata ai microservizi per consentire una maggiore scalabilità, riusabilità e affidabilità dei processi, disaccoppiando e

semplificando i grandi silos tecnologici in moduli più facili da integrare e scalare;

- **Customer Centricity**: Adottare strumenti digitali con architettura headless per offrire nuovi servizi ed esperienze a utenti e dipendenti, standardizzando l'UX sui diversi canali e rendendo trasparenti i complessi flussi di back-end con applicazioni semplici da usare, user friendly e completamente disaccoppiate dai motori di back-end.

Queste linee guida, che verranno riprese poi in dettaglio nei paragrafi successivi del documento, sono state adottate per estensione anche nell'ambito dell'iniziativa di ammodernamento, ossia la trasformazione del parco applicativo as-is, che è in corso e che terminerà nel 2026. L'obiettivo di questa iniziativa sarà quello di ripensare le nuove applicazioni in ottica di utente e processo, di valorizzare i perimetri dei dati coinvolti e di costruire architetture cloud-ready.
Contestualmente a tale attività è prevista un'azione di decomissioning di parte del software di base in uso attualmente in ASPI seguendo logiche di efficientamento dei processi di governance, di riduzione dei costi operativi attraverso economie di esperienza su un minor numero di tecnologie e software di base.

Di seguito si riporta la ***target state architecture*** che rappresenta il disegno complessivo a cui tendere nello sviluppo e trasformazione di, rispettivamente, nuove iniziative o applicazioni legacy.

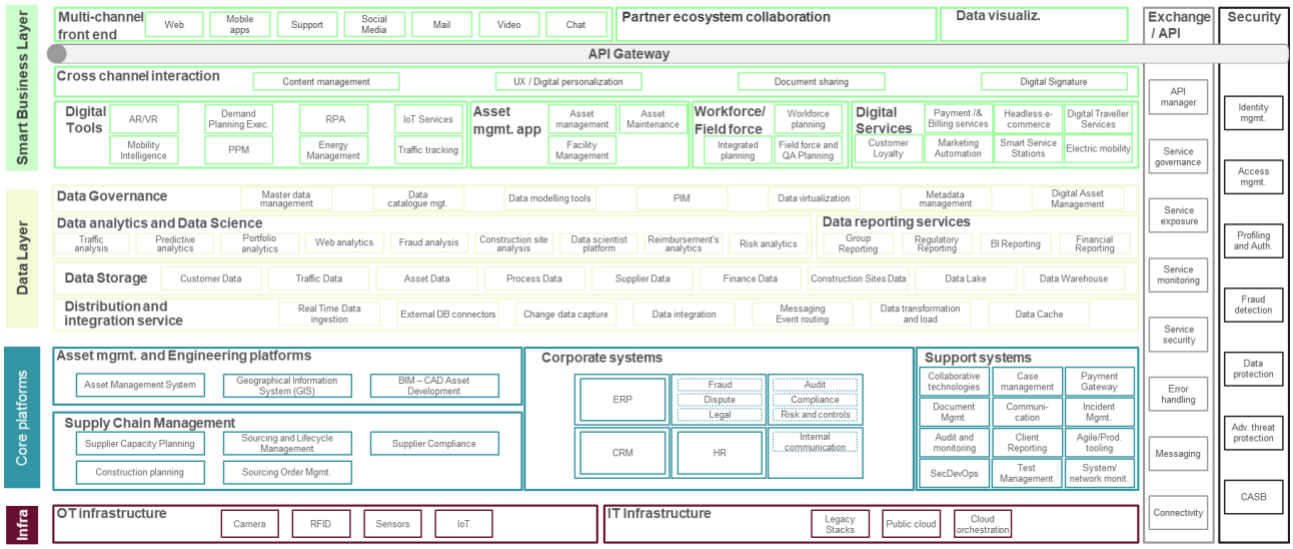


Figura 1. Target state architecture

Analizzando i vari livelli della figura, dal basso verso l'alto, della figura, si evidenziano specifici *layer*:

- quello infrastrutturale (*IT Infrastructure, OT Infrastructure*), che segue un approccio mirato ad una realizzazione delle nuove soluzioni ed alla trasformazione della attuali su paradigma IaaS (Infrastructure As A Service) e che veda una maggiore integrazione della parte "OT" ossia di *Operation Technology* a seguito dell'aumento della diffusione e dell'uso di dispositivi su campo (IIoT – Industrial Internet Of Things) come fonti essenziali di dati per il governo delle infrastrutture, degli impianti e leva di sviluppo dei processi di *smart mobility*;

- quello delle piattaforme principali (*core platform*) che è definito nell'ottica di raggruppare gli elementi abilitanti e acceleratori delle iniziative digitali e supporto dei processi di business;

- quello dei dati e di tutti gli strumenti di gestione e governo degli stessi (*data layer*) sui quali si appoggeranno le soluzioni di *data analytics* e *data science*;

- quello della digitalizzazione dei processi core e non (*smart business layer*) su cui verranno sviluppati *vertical* applicativi per supportare i processi e le attività delle business unit di ASPI.

3 I processi di governo di Enterprise Architecture

Da un punto di vista di governance si ritiene utile riportare e riassumere in questo paragrafo la responsabilità della struttura di IT Enterprise Architecture (di seguito IEA) nei progetti IT di ASPI e delle sue controllate come struttura di governance di gruppo (si veda la Procedura Gestionale di Gruppo – Modello per l'adozione e gestione delle soluzioni IT di Gruppo).

In particolare, la governance architetturale si pone l'obiettivo di assicurare l'aderenza agli standard tecnologici di ASPI, di tutti i progetti che mirino a realizzare soluzioni IT. Tale governance deve coprire tutte le fasi di sviluppo nell'ottica di garantire l'allineamento con la struttura di IEA sin dalle prime fasi e minimizzare eventuali rework o rallentamenti in fase di realizzazione della soluzione. Inoltre, la razionalizzazione del parco applicativo/tecnologico di ASPI su piattaforme corporate, tra i principali obiettivi del programma di ammodernamento, è raggiungibile solo se gli stream applicativi adottano le linee guida IEA limitando la proliferazione di applicativi o sviluppi custom verticali non necessari.

Per assicurare il coinvolgimento nel processo e supportare gli ambiti di sviluppo, per ogni progetto verrà messo a disposizione, su richiesta, un referente della struttura IEA che agirà come punto di contatto del gruppo di sviluppo verso la struttura di architetture di ASPI. Il referente potrà essere richiesto a mezzo mail all'indirizzo mail: enterprise.architecture@autostrade.it .

Il coinvolgimento di tale referente consentirà al gruppo di lavoro di formalizzare due deliverable architetturali:

- **un'architettura di alto livello** che provvederà a mappare tecnologie e funzionalità utilizzate rispetto alle tecnologie a disposizione in ASPI e che dovrà essere nota in fase di avvio del progetto/studio di fattibilità per progetti waterfall e in fase di inception in caso di progetti agile;

- **un'architettura di dettaglio** che prevederà, al minimo, sia un *component diagram*, sia un *flow diagram* da consegnarsi al termine del task di definizione architetturale per progetti waterfall e come deliverable del primo sprint (prime due settimane di avvio del progetto) nel caso di progetti agile.

Questi deliverable verranno poi formalizzati nel blueprint architetturale (si veda paragrafo **Error! Reference source not found.**)

In termini di responsabilità, il referente di IEA dovrà occuparsi di:

- proporre o validare l'architettura dell'iniziativa, tenendo conto dei requisiti tecnico-funzionali;
- chiarire i dubbi architetturali sui quali viene ingaggiato dal gruppo di lavoro;
- validare formalmente le architetture proposte e condivise;
- allineare i gruppi di lavoro con cui collabora rispetto ad evoluzioni rilevanti delle soluzioni tecnologiche in uso;
- facilitare l'ingaggio delle strutture di esercizio da parte del gruppo di lavoro di sviluppo;

Al termine della fase di definizione architetturale, la validazione dell'architettura verrà formalmente notificata attraverso la produzione di documentazione specifica e ingaggio delle strutture di esercizio, principalmente IT Operations. La documentazione specifica riguarderà al minimo il rilascio di un blueprint architetturale, condiviso con CISO, che riporterà al suo interno tutte le informazioni riguardanti l'architettura del sistema in ambito e i flussi interni ed esterni per tenere traccia delle integrazioni. Il documento non conterrà dettagli riguardo l'ambito infrastrutturale (es: numero di server, dimensionamento, indirizzi IP o numero di landscape) perché tali dettagli sono di responsabilità di IT Operations e trovano dettaglio nella documentazione di esercizio relativa a ciascuna applicazione.
Nel caso di cambiamenti dell'architettura in termini evolutivi su singolo componente (es: numero istanze ec2 o upgrade modulo sw di base) la responsabilità di aggiornare del documento sarà lasciata ad IT Operations, mentre nel caso di revisione dell'architettura complessiva (es: introduzione di un nuovo servizio o nuova integrazione) sarà la struttura di Enterprise Architecture ad aggiornare il documento.

3.1 Blueprint architetturale

La creazione di documentazione tecnica che sia aggiornata e disponibile è critica per diversi fattori, che spaziano dall'operation alla cybersecurity, ma includono anche aspetti di supporto allo sviluppo. Mantenere una documentazione aggiornata è fondamentale per la gestione dei passaggi di consegna, per evitare il *reverse engineering* e assicurare un ciclo di vita documentato e governato ad un'architettura IT.
La struttura CTO/Architetture con la struttura CISO hanno congiuntamente definito un template documentale che si trova al seguente link:

https://dwisrv.cto-utilities.prod.aws.autostrade.it/en/Public/wiki-security/abb-repository/documentation-abb

La compilazione di questo template è funzionale:

- alla governance delle architetture (responsabilità CTO/IEA);
- alla fase di implementazione delle infrastrutture IT a supporto e alla gestione della fase di esercizio IT (responsabilità CTO/ITO);

- alla valutazione dei rischi di cybersicurezza, alla definizione e implementazione delle logiche di sicurezza IT adeguate e per la gestione della risposta agli incidenti di sicurezza.

Nel caso si adottino cicli di sviluppo incrementali, è richiesto che ad ogni incremento si verifichi la congruità della documentazione con la soluzione IT che verrà rilasciata, limitando il più possibile il rischio di discrepanze.
In questo senso si suggerisce:

- che il RIA di un progetto si assicuri che la documentazione sia in draft fino ad ogni major release, ovvero la documentazione dovrebbe essere rivista e allineata ad ogni fine sprint, anche in caso di mancanza di una major release;
- di seguire lo standard in modo che la documentazione prodotta sia di facile utilizzo;
- le informazioni contenute devono essere effettivamente necessarie allo scopo di mantenere la conoscenza dell'applicazione.

La struttura di tale documento dovrà prevedere, al minimo:

1. Descrizione del progetto, contenente informazioni come il codice applicativo, o il product owner;
2. Descrizione use cases, contenente le informazioni sui casi d'uso implementati
3. Interazione fra i sistemi, che descrive come l'applicazione interagisce all'interno fra i suoi moduli, e con l'ambiente esterno
4. Dettaglio dei flussi, che descrive i flussi di comunicazione
5. Disponibilità e Scalabilità, che descrive come la disponibilità e la scalabilità sono raggiunte.

4 Linee guida di disegno architetturale e buone pratiche

Nell'analizzare le linee guida architetturali e nel suggerire le buone pratiche ci si deve porre nell'ottica che le applicazioni moderne necessitano la valutazione di un'architettura che prenda in considerazione gli aspetti di processo (tipologia di processo, task e attori) e il dominio funzionale dei dati di business.

In questo senso si suggerirà un modus operandi che ha l'obiettivo di indirizzare la definizione di un'architettura IT secondo le tre dimensioni: quella di processo, quella del dato e quella delle modalità realizzative.

4.1 La dimensione del processo

Parlando di processi, li definiamo primari o "core" se sono processi legati al business principale dell'azienda e sono costituiti da attività generatrici di valore. I restanti processi riguardano processi cosiddetti di supporto essendo essi costituiti da attività che consentono l'esecuzione di task non core e che garantiscono l'adeguato supporto ai processi primari aumentandone efficacia ed efficienza.

Nel caso di applicazioni realizzate per Autostrade per l'Italia è opportuno fare riferimento alla modellizzazione del **sistema Pway** che indica con 38 macroprocessi (livello L0) altrettanti domini

funzionali di business di riferimento, Questi 38 sono poi differenziati tra i processi di business, ossia quelli “core”, e quelli di supporto e management ossia “non-core”.

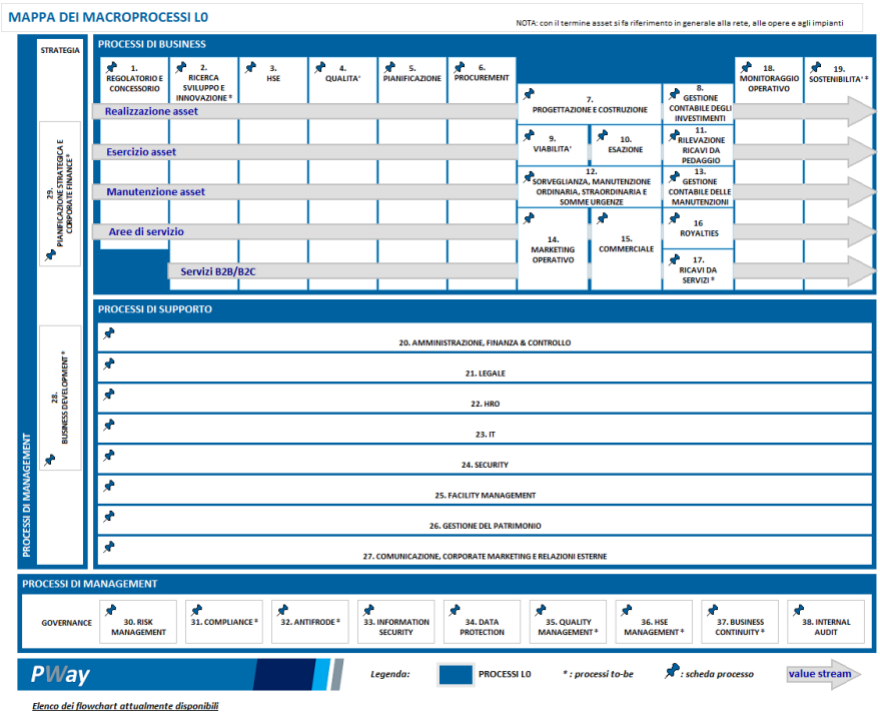


Figura 2 - Processi Pway

Questa prima differenziazione, ossia il supporto che il sistema IT in esame fornirà a processi core e non-core, consente già di indirizzare una prima scelta architetturale, che coinvolgerà l’uso:

- **di servizi o di piattaforme (SaaS o PaaS)** per i processi di supporto, per scenari che richiedano adeguamento a standard di mercato o per normalizzazione di processi che presentavano prima della loro trasformazione (piano di trasformazione parco applicativo onpremise) specificità non giustificate (si veda per approfondimento 4.5);

- **di infrastruttura cloud (IaaS)** nel caso di scenari che presentino nella loro specificità un elemento di generazione di valore, che non presentino analoghe soluzioni presenti a mercato o che siano relativi a scenari di innovazione o di ricerca e sviluppo;

- **di infrastruttura on-premise** nel caso di scenari che presentino nella loro specificità l’appartenenza ad un perimetro di servizi critici di Autostrade per l’Italia come operatore di servizi essenziali (OSE) che evidenzino dei vincoli (si veda IEC 62433) che ne richiedano l’installazione sui datacenter di Autostrade per l’Italia o che, nel caso di soluzioni che prevedano l’uso di componenti di *operation technology* (OT), siano realizzati “on edge”.

Nel caso un’iniziativa si collochi all’interno di processi, le cui soluzioni o piattaforme IT siano già state individuate, la scelta tecnologica di tale iniziativa, salvo rare eccezioni, va in continuità con

questa scelta per favorire il riuso di piattaforme e razionalizzare gli aspetti di integrazione tra sistemi che supportino lo stesso processo di business.
In questo senso, in ottica di Gruppo, oltre alle “tech foundations”, ai “sistemi enterprise” e ai “sistemi di business comune” (si veda la Procedura Gestionale di Gruppo – Modello per l’adozione e gestione delle soluzioni IT di Gruppo) sono raccomandate sinergie tra le Società del Gruppo anche nei sistemi di “business specifici” laddove possibile.

4.2 La dimensione del dato

Sul fronte dei dati è opportuno concentrare l'attenzione sui modelli di dominio logici dell'applicazione e sui dati correlati con l’obiettivo di individuare sotto-domini indipendenti di dati e contesti diversi all'interno della stessa applicazione (si vedano i concetti di bounded context e domain driven design approfonditi in 5.1.5). L’analisi del caso d’uso determinerà la scelta della soluzione più opportuna per la gestione del dato:

- **database operazionali**, nel caso di soluzioni che prevedano una grossa movimentazione dei dati prevedendo l’uso di strutture relazionali (DBMS) nel caso di dati strutturati e prevedendo strutture più flessibili (NoSQL database) nel caso di dati semistrutturati o non strutturati. L’uso di queste strutture è anche indicato per scenari *realtime* o *near-realtime*

- **datawarehouse**, nel caso di soluzioni che mirino alla creazione di vaste banche dati denormalizzate con alta profondità temporale, che facciano uso di attività di ETL e di data preparation per realizzare domini di dati (datamart) per finalità di standard reporting o self-service reporting;

- **datalake**, nel caso di soluzioni che mirino alla creazione di aree di dati eterogenei (sia in termini di provenienza, sia in termini di tipologia) per effettuare analisi euristiche da parte di data scientist o per attività di machine learning volte alla costruzione di modelli predittivi e di AI da riportare in ambito operazionale;

- **lakehouse**, come evoluzione e superamento dei limiti delle soluzioni di datawarehouse e datalake, con l’obiettivo di avere una piattaforma unificata per la raccolta e l’analisi di dati eterogenei, strutturati e non.

4.3 Le modalità realizzative

In merito alla tipologia di realizzazione e al perimetro di applicazione, si scoraggia un approccio che miri a coprire un processo *end-to-end* o parte di un processo attraverso la realizzazione di un cosidetto *vertical* o “monolite”. Queste soluzioni software che ambiscono a supportare un intero processo di business includendo sia gli elementi frontend sia tutte le logiche di business di backend, comportano un articolato strato di business logic che fa uso di complesse strutture dati. Tale approccio, pur rappresentando una notevole semplificazione architetturale, presenta, nel caso di attività evolutive o di cambi in termini di processo, forti criticità sia in termini di gestione e

manutenzione del codice (alto rischio di refactoring e regressioni) sia in termini di esercibilità della soluzione.

Al contrario, in ASPI, grazie anche all'adozione di modalità di sviluppo agile che consentono sviluppi incrementali, si favorisce la concettualizzazione di architetture modulari fino all'uso di microservizi con l'obiettivo di costruire sistemi che garantiscano diverse geometrie di scalabilità assicurando flessibilità e resilienza e che presentino bassi accoppiamenti tra i moduli che implementano le logiche di business.

In questo senso, rispetto alle logiche di funzionamento, proprio in un'ottica di indipendenza di un modulo rispetto ad un altro, si suggerisce l'adozione di logiche di architetture ad evento (*event driven*) piuttosto di quelle basate sulla transazionalità (*request driven*) che peraltro non sono raccomandabili nel paradigma a microservizi.
Un'architettura guidata dagli eventi può aiutare nella realizzazione di un sistema flessibile che consenta di poter effettuare decisioni di business in tempo reale. Questo tipo di paradigma è molto frequente negli scenari dell'Internet of Things (IoT), in cui produttori e consumatori dell'informazione condividono lo stato e le informazioni della risposta in tempo reale. Inoltre, un'architettura guidata dagli eventi consente di migliorare la scalabilità e i tempi di risposta delle applicazioni nonché l'accesso ai dati e il contesto su cui far poggiare il processo decisionale.

Lato infrastrutturale, le precedenti logiche di consolidamento e aggregazione che miravano a ridurre i costi operativi di esercizio delle varie soluzioni tecnologiche comuni (es: host di consolidamento degli application server della stessa tipologia) si sono trasformate in logiche di aggregazione che mirino al riuso di servizi applicativi con l'obiettivo di ridurre la replicazione di moduli similari all'interno del Sistema Informativo (uso di servizi/microservizi) e assicurare al contempo un migliore controllo su ownership di funzioni comuni, di alcuni tipi di dati e loro utilizzo.
La necessità poi di portabilità del software ha spinto verso la realizzazione di soluzioni basate su tecnologie di virtualizzazione e di abbandono delle soluzioni basate su server fisici (anche per i database), che si è spinta fino all'uso di container e relativi orchestratori (e.g. *kubernetes*) nel caso di servizi con basso footprint o dell'adozione di un paradigma architetturale a microservizi.
Lato dati, l'adozione di pattern architetturali orientati all'autonomia del singolo modulo software (e indirettamente la maggiore libertà del gruppo di sviluppo in termini di minor dipendenze funzionali esterne) ha dato luogo ad un aumento del numero di *datastore* e conseguentemente alle necessità di gestione e di governance degli stessi.

La modularità, la frammentazione e la ditribuzione dei vari moduli costituenti una soluzione IT ha poi reso fondamentale definire, in termini di comunicazione tra sistemi diversi, le possibili strategie di integrazione (si veda paragrafo 4.8) sfruttando *layer* di integrazione comuni anche in ottica di facilitare la *data governance* e le logiche di democratizzazione del dato.

Nei paragrafi successivi molti dei concetti qui espressi in maniera sintetica verranno ripresi e declinati rispetto alle specifiche linee guida espresse all'inizio di tale paragrafo.

4.4 DevSecOps

Rispetto al concetto più noto e diffuso di DevOps, ossia la metodologia di sviluppo che vede sempre più coinvolti i team di sviluppo (Development) e di esercizio (Operation), vista la crescente

importanza delle tematiche di sicurezza il termine DevSecOps esprime la necessità di passare da un vecchio concetto di verifica della sicurezza dopo le fasi di sviluppo, che magari veniva affidato ad un team dedicato, ad uno in cui gli aspetti di sicurezza e esecuzione dei test sul codice prodotto rientrano nelle dinamiche dello sviluppo nel cosiddetto approccio “shift left”.

Come ulteriore conferma della bontà dell’approccio, l’attivazione delle filiere di DevSecOps sulle principali piattaforme ha già consentito nella prima introduzione fatta in ASPI una riduzione dei tempi di rilascio (fino al 90%) e un aumento della frequenza di rilascio della parte di change e fix, introducendo la possibilità di rilascio incrementale delle nuove funzionalità dell'applicativo.

In questo senso ASPI prevede come architettura a supporto del processo, l’utilizzo di una pipeline che partendo da un repository comune dei sorgenti prevede procedure automatiche sia per le applicazioni custom (IaaS o onprem), sia per le piattaforme utilizzate in ASPI. La definizione di queste pipeline che copre il processo di sviluppo end-to-end su tutto il landscape, consente sia una facilitazione per i tema di sviluppo, sia una maggiore governance del processo che beneficia dell’integrazione con i processi di test automation legati all’ambito della sicurezza (SAST e DAST), della qualità del codice rilasciato e della corretta esecuzione di test funzionali e di integrazione.

Rispetto agli standard ASPI, di norma vengono messi a disposizione landscape a tre ambienti e più raramente a quattro ambienti nel caso di esigenze specifiche. Un tipico landscape a quattro ambienti è costituito da:

- **Sviluppo** – necessario per valutazioni di fattibilità e primi sviluppi, definizione del primo livello dello schema dei dati, unit e functional test. Questo ambiente è consegnato ai gruppi di sviluppo;

- **Test** – anche detto ambiente di integrazione, qui vengono eseguite le attività relative a *integration and system testing* dell’applicazione e di norma si realizza la prima versione candidata ad essere portata in esercizio;

- **Quality Assurance** – anche detto ambiente di User Acceptance Test (UAT), su questo ambiente si svolgono i test utente e i test end-to-end. In alcuni casi questo ambiente usa data set anonimizzati dell’ambiente di produzione.

- **Produzione** – è l’ambiente di esercizio dell’applicazione.

Eccezionalmente può essere previsto un ambiente aggiuntivo, denominato di “hot fix” viene inserito tra quello di Quality Assurance e quello di Produzione, e viene utilizzato per la gestione dei bug in produzione e rapida risoluzione attraverso la replicazione del problema in tale ambiente e successiva realizzazione del *workaround*. Tale pratica garantisce una più rapida risoluzione della *issue* in attesa che ne venga introdotta la correttiva nel ciclo di sviluppo successivo.

4.5 L’approccio platform first

L’approccio *platform first* nasce principalmente dall’esigenza di indirizzare esigenze di sviluppo verso soluzioni già disponibili e che garantiscano in termini di processo un adeguata compliance con standard specifici ed esigenze di mercato piuttosto che tempi di realizzazione più rapidi. L’uso

di tali piattaforme, come già anticipato, è particolarmente indicato nell'ambito dei processi di supporto (non-core), ove la specificità di un processo o di una realizzazione custom dello stesso non porta valore ma anzi aumenta i costi operativi limitando efficacia ed efficienza.

In tal senso l'uso di tali piattaforme favorisce:

- la rapidità di realizzazione di soluzioni software;
- la possibilità di realizzare una soluzione che rispetti già le aspettative del cliente (perché basata sull'esperienza del vendor in casi d'uso analoghi per esigenze similari anche se di industry diverse) e che sia in linea con eventuali standard o specifici requisiti (es: una soluzione PaaS di CRM (Customer Relationship Manager) offre già di default la compliance ai requisiti in ambito di privacy come il GDPR);
- la modificabilità di tale realizzazione, nel caso di PaaS che utilizza paradigma *low code* si arriva quasi ad una programmazione dichiarativa facilitando la condivisione del processo con il business;
- la facilità di integrare un processo con altri realizzati all'interno della stessa piattaforma usando gli stessi dati;
- la disponibilità di funzionalità OOB (out-of-the-box) sia in termini funzionali che di integrazione rispetto ad altre piattaforme o altri software di terze parti.

Si riportano di seguito le principali piattaforme PaaS/SaaS (Platform As A Service, Software As A service) in uso in ASPI e il loro utilizzo e perimetro di processo:

- **Microsoft 365**, per le soluzioni legate all'office automation;
- **SAP S4 HANA**, per lo sviluppo di funzionalità legate a processi in ambito amministrativo/contabile;
- **Salesforce**, per lo sviluppo di soluzioni per la gestione dei clienti / utenti esterni (CRM – Customer Relationship Management) e di workforce management;
- **SAP Success Factor**, per lo sviluppo di processi in ambito organizzativo del personale interno;
- **Hyland Onbase**, per le funzionalità in ambito documentale;
- **ServiceNow**, per lo sviluppo di soluzioni legate alle funzionalità della gestione di workflow e pianificazioni per utenti/processi interni e i processi ITSM

4.6 L'approccio cloud first

Un approccio "cloud first" comporta il ripensamento del software e delle architetture a supporto nella logica del *cloud readiness* ossia di quanto le nuove architetture possano avvalersi delle nuove tecnologie e servizi disponibili nel paradigma cloud per aumentare e accelerare, con la loro realizzazione, la produttività dei processi di business che supportano.

Autostrade per l'Italia ha scelto come cloud provider per la parte IaaS (Infrastructure As A Service) Amazon Web Services (AWS) e Microsoft Azure, dove si stanno implementando tutte le nuove architetture delle nuove applicazioni. Su questi cloud provider saranno portate progressivamente la maggior parte delle architetture e applicazioni attualmente *on premises* installate nei datacenter di ASPI nell'ambito del piano di trasformazione citato in precedenza in coerenza con le scelte definite in 4.1.
Per le attività di governance del cloud ASPI ha istituito un CCoE (Cloud Center of Excellence) che rappresenta il gruppo di lavoro di riferimento per l'emanazione degli standard e degli strumenti di governance del cloud.

Nello specifico perché un'applicazione possa dirsi "cloud ready" o se sviluppata nelle nuove logiche, definirsi "cloud native", deve prevedere delle caratteristiche che prevedano lo sviluppo dell'applicazione con logiche di containerizzazione, di sviluppo in microservizi e di una loro gestione con processi e strumenti *DevSecOps*.
In questo contesto lo sviluppo delle nuove applicazioni passa anche dall'uso di FaaS (Function-as-a-Service) per:

- indirizzare servizi già disponibili presso il Cloud Provider che non è conveniente sviluppare da zero;
- sviluppare logiche di business *serverless* che consentano un uso della risorsa computazionale basata su logiche ad evento e non continua secondo l'approccio tradizionale;

Nell'adottare un approccio cloud first, ASPI non intende però vincolarsi ad uno specifico cloud provider, ma capitalizzare le esperienze per realizzare un sistema informativo multi-cloud che, nella scelta delle tecnologie, segua lo spirito della cloud readiness ma che ponga specifica attenzione a possibili elementi di lock-in.
In tal senso e in particolare sui processi delle aree di "core business" o su quelli che caratterizzano ASPI come OSE si intende mantenere una indipendenza dal fornitore IaaS ricorrendo, per l'implementazione dei servizi caratteristici cloud native, a soluzioni di terze parti che sono pienamente compatibili con i maggiori cloud provider del mercato.

A questo proposito si elencano di seguito le scelte tecnologiche che ASPI ha previsto in tal senso:

- la soluzione di API Management è quella di **IBM API Connect**;
- la soluzione di Pub/Sub è realizzata su **Confluent Kafka**;
- la soluzione di DevOps è realizzata attraverso tecnologie di diversi produttori: **Gitlab, Nexus Sonatype, Jenkins, IBM Urban Code Deploy, HCL AppScan**);

- la piattaforma di schedulazione è **BMC Control-M**;
- la piattaforma di monitoraggio con **Zabbix**, **AWS Cloudwatch** , **Azure Monitoring** e APM con **Dynatrace**

Con questo approccio, le tecnologie scelte, su AWS e MS Azure sono state selezionate facendo delle valutazioni che consentissero da una parte una forte facilitazione nello sviluppo e nella gestione dell'esercizio di una soluzione ma che presentassero anche una *way-out* tecnologica su prodotti equivalenti. Si riporta di seguito alcuni esempi:

- la parte DBMS è principalmente su soluzioni gestire **RDS Aurora basate su PostgreSQL** che consentono un porting su tecnologia PostgreSQL se necessario;
- la parte OS è **RedHat Enterprise Linux** all'interno degli host EC2 (Elastic Cloud Computing) forniti;
- la soluzione della JVM (java virtual machine) è **AWS Corretto** che è un *fork* del progetto di openJDK con supporto di lungo termine (LTS, long term support);
- l'utilizzo di **AWS lambda**, **AWS batch**, **Azure Function** viene favorito per la gestione serverless ma anche per l'indipendenza di tale soluzione dal linguaggio con cui viene scritto il codice (il codice può essere java e quindi portabile su soluzioni FaaS di altri cloud provider);
- l'utilizzo di **AWS EKS e Azure AKS** soluzioni di containerizzazione compatibile con gli standard k8s. L'uso di AWS EKS verrà implementato per i processi non core e non business critical.

Di contro, non viene invece fatto uso di tecnologie proprietarie che vincolerebbero le architetture a poter essere ospitate solo nello IaaS di AWS o Azure. Questo approccio è scoraggiato anche su processi non core.

Rispetto alle tecnologie serverless, si ritiene importante portare un punto di attenzione rispetto ai FaaS (Function as a Service) come AWS lambda, AWS Batch e AWS Glue. Questi servizi si basano su logiche che:

- consentono di disaccoppiare la funzione applicativa sviluppata dall'infrastruttura sottostante;
- presentano una maggiore facilità di deployment rispetto all'approccio classico (uso di EC2);
- consentono di garantire alta scalabilità per la funzione invocata.
- consentono una spesa legata all'effettivo uso del servizio seguendo la variabile del numero di invocazioni.

Di contro però, l'uso di tali strumenti va ben ponderato rispetto agli elementi seguenti:

- la necessità che il servizio abbia latenze basse e quindi si debba confrontare, nel caso di FaaS con tempi di "warm-up" che non rispettino quel vincolo di tempo;
- gli eventi che generino le invocazioni non siano prevedibili *ex ante* (ad es: siano legati a variabili esogene come richieste utente di un frontend su public internet);

- gli eventi che generano l'invocazione di tali servizi presentino una frequenza tale che risulti più conveniente una realizzazione della stessa funzione in modalità classica (con uso di server dedicato).

Gli ultimi due punti sono molto importanti nella definizione di un'architettura in cui si preveda l'uso di FaaS, proprio per non vanificare la sostenibilità economica della stessa.
Provando a portare un esempio: la realizzazione di un frontend utente di una piattaforma e-commerce con AWS lambda se da una parte garantirebbe una forte scalabilità (seppur con tempi di warm-up da valutare), legherebbe la variabile del numero di utenti sia ai ricavi (più utenti ho sulla piattaforma, più presumibilmente ho acquisti effettuati) sia ai costi che quel servizio genererebbe (il pagamento di AWS Lambda sarebbe legato direttamente al numero di utenti che fanno uso della piattaforma).

Sull'uso e le buone pratiche nell'utilizzo della piattaforma, ci si riferisce, come detto in precedenza, ai lavori del CCoE i cui output, se d'impatto architetturale, verranno riportati in questo documento negli aggiornamenti successivi.
Si faccia poi riferimento nel caso di sviluppo di architetture su AWS e Azure al documento di Well-Architected Framework.

4.7 L'approccio data centric

Come già detto in precedenza, le architetture moderne riportano il concetto di processo e di dato al centro.
In questo senso il dato necessita di essere rivalutato rispetto all'approccio tradizionale:

- in un'ottica del dato funzionale ad un processo (input e output di processo di business);
- in una logica di possibile ruolo maggiormente trasversale dello stesso (per uscire da un ruolo unicamente vincolato ad un silos);
- in una logica strategica come elemento su cui basare le decisioni (KPI, Key Performance Indicator) ed efficientare processi nell'ottica di ridurre i costi operativi;
- in una logica di democratizzazione, per valorizzarlo in una logica di trasparenza, perché il dato sia accessibile internamente e diventi elemento di dialogo con il business per estrarre elementi a valore.

In ASPI è stato realizzato un *data layer* che consenta di poter essere un abilitatore degli elementi citati sopra in una strategia di valorizzazione del dato nell'ambito del business dell'Azienda e che porti conseguentemente ASPI ad essere un'azienda "data driven".

Le tecnologie usate all'interno di tale data layer sono le seguenti:

- **AWS S3** - "Simple Cloud Storage Service" è lo storage selezionato per la conservazione dai dati, questo rappresenta il vero e proprio D-LAKE;

- **AWS Glue** è utilizzato come servizio per lo sviluppo degli ETL in modalità "Spark". Legato a Glue ci sono il Data Catalog e i Crawlers. Il primo, a partire dal Job, permette di tracciare il modello dati e il relativo flusso, il secondo permette, volendo, di "deployare" automaticamente il modello dati in Amazon RedShift;
- **AWS EMR** è utilizzata come piattaforma per elaborare, analizzare e applicare il machine learning (ML) velocemente ai Big Data, utilizzando framework open-source;
- **AWS Lambda** è un servizio serverless per l'esecuzione di codice (python o spark) custom per esigenze specifiche non coperte direttamente da Glue;
- **Amazon Athena** è un servizio che permette di interrogare i dati conservati in S3 in modalità SQL-Like;
- **AWS Storage Gateway** che permette di estendere S3 come file system remoto sui sistemi on premise;
- **AWS CloudFormation** per il change management di infrastruttura esteso in alcune funzionalità con la aws cli
- **Amazon APPflow**, ossia il servizio di AWS per integrazioni API sorgenti SaaS esterne (es. Salesforce, Dynatrace, ServiceNow);
- **Tibco Spotfire** è un software SelfService per la creazione di dashboard utente in ambito *data visualization* e *self-service reporting*.
- **Tibco Data Virtualization** è un software per implementare un layer semantico e una prima pre-aggregazione per l'analisi dei dati. Può essere utilizzato anche per l'esposizione di metadati o dati master (Master Data) via oData e/o API.

L'architettura prevede le seguenti aree:

- **Tier 0 (Raw Data Zone)**: è l'area destinata ad accogliere i dati in ingresso nella loro forma sorgente, senza modifiche rispetto al tracciato di ingresso, a meno dei campi tecnici descritti successivamente. Tipicamente accoglie pacchetti di informazioni che rappresentano variazioni. Il meccanismo di acquisizione processa i pacchetti disponibili nel Tier 0 per alimentare il Tier 1. Gli incrementi acquisiti non vengono eliminati per garantire la completa ricostruibilità del Tier 1.
- **Tier 1 (Datalake):** Il Tier 1 è la struttura che ospita il vero e proprio 'Data Lake'. Il Data Lake conserva informazioni eterogenee, allo scopo di consentire una veloce fruizione sia per alimentare il Tier 2 (livello analitico) sia allo scopo di supportare i bisogni informativi dei Citizen Data Scientists e del Business, attraverso query dirette o collegamento a Data Science Tools od Advanced Analytics Tools. Il Tier 1 sarà interrogabile attraverso strumenti sql-like oppure attraverso tools connessi direttamente al repository S3.
  Il Tier 1 contiene le informazioni acquisite dal Tier 0. Ai fini della certificazione del dato è raccomandata la modellazione "on read" del dato, e in particolare:

  - la definizione della primary key
  - la definizione delle chiavi univoche aggiuntive
  - obbligatorietà delle colonne
  - la definizione delle relazioni
  - il controllo applicativo dell'integrità referenziale.

  In fase di analisi si definisce il livello di gestione delle anomalie (warning, errore, modalità di segnalazione).

Qualora alla sorgente sia già controllata l'integrità referenziale tra due tabelle, e queste vengano acquisite nell'ambito dello stesso flusso di acquisizione, è possibile non ripetere il controllo nel T1.
In questo modo il Data Lake è in grado di supportare gli use case di bisogni non noti a priori e della sperimentazione, quindi non perdendo informazione nella trasformazione. È inoltre in grado di alimentare il Tier 2, attraverso trasformazioni anche di struttura volte ad ottimizzare per la specifica funzione analitica richiesta. Inoltre il Tier 1 è completamente ricomputabile dal Tier 0 ed è implementato attraverso Object Storage S3.
In Tier 1 è presente, inoltre, un processo di controllo della qualità del dato implementato attraverso il framework AWS DeeQu.

- **Tier 2 (Analytical Data Zone)**: Il Tier 2, o livello analitico, contiene dati in strutture costruite ad-hoc per supportare funzioni analitiche, ottimizzate per lo specifico uso (Data Mart, modelli DWH, …).
  Il Tier 2 supporta lo use case relativo a bisogni noti, derivati eventualmente dalle sperimentazioni condotte dai Data Scientist e dal Business sul Tier 1 e frutto di analisi specifiche volte alla realizzazione di reports, dashboards, algoritmi etc. La modellazione dei dati nel Tier 2 è specifica dello use case ed anche dello strumento con cui saranno fruiti, tra le varie ottimizzazioni da considerare la riduzione della profondità storica, con una precisa finalizzazione allo use case. Il Tier 2 è completamente ricomputabile dal Tier 1.
  In ottica evolutiva il Tier 2 potrà ospitare anche modellazioni più raffinate e generali dei dati, una volta che i prototipi realizzati nelle fasi di sperimentazione abbiano raggiunto un livello di maturità adeguato.
  Il Tier 2 è implementato attraverso Database AWS Redshift.

Nell'ambito di disciplinare l'uso di questo da layer che è previsto che nel corso del 2025 evolva in una soluzione di lakehouse è opportuno precisare che:

- il data layer nella sua accezione di "datalake" non può e non deve essere utilizzato come "data hub" sfruttando il fatto che su tale layer vengono persistiti la stragrande maggioranza dei dati aziendali;
- il data layer non è indicato per realizzare reportistiche operative che devono essere realizzate negli ambienti operazionali;
- il data layer non può e non deve essere utilizzato per esigenze di storicizzazione. Tali esigenze devono poter essere gestite attraverso l'uso di soluzioni altamente capacitive a basso costo.

4.8 Linee guida di integrazione

Questo paragrafo si pone come obiettivo quello di definire delle linee guida nella definizione delle strategie di integrazione tra moduli, applicazioni e piattaforme software. Questa si propone come linea guida per l'implementazione di integrazioni tra applicazioni, moduli e piattaforme nell'ambito del Sistema Informativo di Autostrade per l'Italia.

Nella definizione delle linee guida d'integrazione si fa riferimento agli obiettivi e i lavori del team "Design Authority – Architetture" del processo di Digital Transformation di ASPI. Nello specifico, si raccomanda, laddove possibile, di implementare per le interfacce un approccio *API first* disaccoppiando i diversi livelli e sistemi attraverso strumenti di integrazione per consentire una

standardizzazione e democratizzazione dei dati, così come la scalabilità e la riusabilità dell'architettura. In questo senso si raccomanda di evitare il “point-to-point”, rendendo l'accesso ai dati agnostico dalle applicazioni attraverso l'esposizione di end-point standardizzati.

Nella definizione di una moderna strategia di integrazione è opportuno tenere in considerazione alcuni elementi di caratterizzazione:

- lo stile d’integrazione;
- la tipologia di informazione e dato che viene scambiato nell’interfaccia d’integrazione;
- i volumi di informazioni coinvolti;
- le modalità di produzione dell’informazione;
- le modalità di consumo dell’informazione;

Nello specifico lo stile d’integrazione è il primo elemento che determina la strategia di integrazione e può essere di tre tipologie:

- **data-centric**: quando l’obiettivo primario è mantenere e aumentare la coerenza dei dati e mantenere un legame tra sistemi appartenenti a domini diversi;

- **event-centric**: quando l’interfaccia deve poter propagare rapidamente eventi di business o un cambio dei dati ad altri sistemi che devono reagire a tale evento o analizzarli;

- **application-centric**: quando si tratta di combinare funzionalità e dati esistenti in nuovi modi per fornire nuovi processi o servizi o per ottimizzare quelli esistenti.

La raccomandazione, in linea con le policy architetturali che richiedono modularità, idempotenza e riuso, è quello di orientarsi su uno stile *event-centric* o *application-centric*. In questo senso, per le nuove integrazioni, l’uso di un’integrazione basata sul dato (es: uso di interfacce DB2DB con uso di viste) non è più consentita e supportata.

Analizzando poi la tipologia di dato, questo determina gli aspetti di sicurezza e a valutazioni sul rischio rispetto all’esigenza di business. Su questo specifico aspetto è opportuno riferirsi ai documenti in ambito sicurezza e ad un passaggio con le strutture deputate per l’analisi dello *use case*. È imprescindibile, in ogni caso, sviluppare l’integrazione usando protocolli sicuri che si basino sull’uso di chiavi, certificati o token riferendosi alle linee guida ASPI (wiki security) e agli standard di mercato. In questo senso è opportuno dare una breve introduzione all’approccio *Zero Trust* che Autostrade ha adottato come postura di sicurezza a cui tendere. Introdotto nel 2010 da John Kindervag, analista di Forrester Research, questo implica un un cambio di strategia da "Fidarsi, ma verificare sempre" a "Verificare sempre, senza fidarsi". Questa strategia si applica ad utenti, sistemi, dispositivi e server che devono essere considerati non attendibili fino a prova contraria; ogni utente, sistema, dispositivo o server viene considerato affidabile per accedere a una risorsa fino a quando non ne vengono verificate l'identità e l'autorizzazione.

Anche l’aspetto legato ai volumi delle informazioni che vengono scambiate, risulta un elemento determinante per fare le corrette valutazioni relativamente ai requisiti di performance della *user experience* e dell’interfaccia M2M (*machine to machine*) coinvolta. Una grossa movimentazione di dati potrebbe richiedere uno stile d’integrazione orientato al dato e non all’evento e potrebbe escludere il paradigma d’integrazione via API. D’altra parte, grosse

movimentazioni di dati, escludendo processi di MDM (*Master Data Management*) e di *Data Lake Ingestion*, potrebbero evidenziare una scarsa chiarezza nell'individuazione della golden source dell'informazione o uno scorretto posizionamento all'interno del dominio funzionale del modulo software che richiede tali dati.

Analizzando poi le modalità di produzione e di consumo dell'informazione, questo elemento determina una riflessione su come sistemi diversi si scambiano i dati e se possa essere più opportuno orientarsi verso un paradigma orientato all'integrazione o più orientato al paradigma API. Quest'ultimo, come detto precedentemente, richiede, da parte di chi produce ed espone l'informazione, una pubblicazione del dato che privilegi la generalizzazione delle informazioni esposte per favorire il riuso da parte delle diverse tipologie di consumatori. Inoltre, le API devono avere una semantica e schemi di controllo delle versioni ben definiti, in modo che gli aggiornamenti non interrompano altri servizi.

Rispetto, infine, al consumo dell'informazione è utile valutare come la business logic dell'applicazione intende utilizzare le informazioni provenienti dall'esterno, scegliendo un approccio *data collect* laddove lo stile d'integrazione sia orientato al dato e *data connect* laddove si vada su uno stile di integrazione che miri ad un arricchimento dell'informazione in una logica di integrazione ad evento.

Si ritiene opportuno poi precisare che, nel caso di applicazioni denominate "ibride" come deployment model, si debba valutare che tipo di integrazioni esistono verso il CSP di riferimento, questo per considerare nella valutazione:

- problematiche di performance (es: latenza)
- problematiche di sicurezza (es: protocolli non sicuri)

Come sintesi e conclusione, si riporta il seguente diagramma di Gartner come supporto riassuntivo alla scelta della strategia di integrazione più indicata.

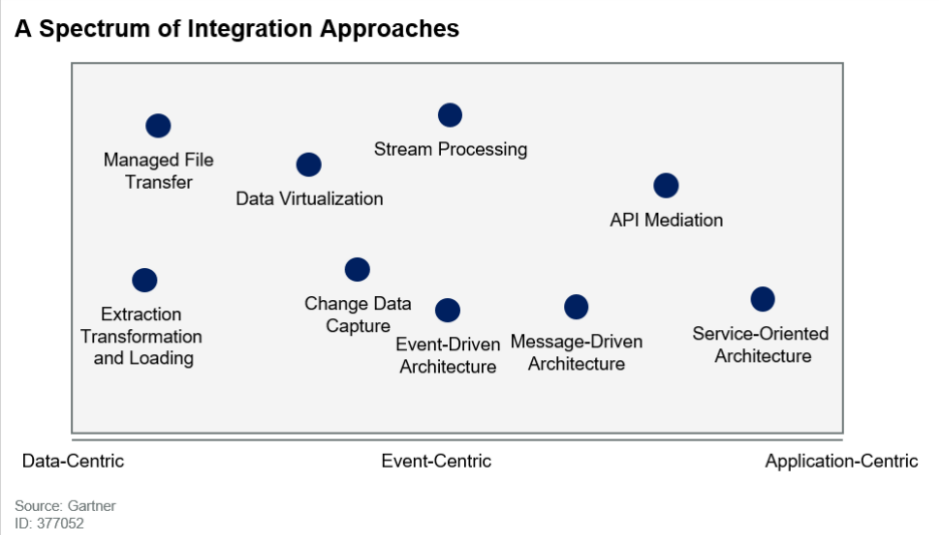


Figura 3 - Stili di integrazione

Nell'ambito degli strumenti a disposizione per l'integrazione riportata nella tabella di seguito i software di base che sono in uso presso ASPI e che rispondono ai diversi stili d'integrazione e alle

diverse caratteristiche delle interfacce che si intende realizzare. Si faccia riferimento all'appendice di questo documento per i livelli software supportati.

Stile di integrazione	Use case	Software ASPI	Protocolli supportati*	Note
Data-Centric	Managed File Transfer	IBM Sterling Integrator	SFTP, FTPS, SMTP, MQ, JMS	Richiesto obbligatoriamente per i trasferimenti file inter-società o con terze parti.
Data-Centric	ETL	AWS Glue	N.A.	Utilizzato per scenari di ETL in cloud
Event-Centric	Change Data Capture	IBM CDC	Proprietario	Use case per change data capture
Event-Centric	API mediation	IBM API Connect, SAP Cloud connector	SOAP, REST	Si raccomanda l'uso di interfacce REST
Event/Application-centric	Message-Driven Architecture	Confluent Kafka	JDBC, MQTT, Websocket, HTTP	Se ne raccomanda l'uso come piattaforma di pub/sub
Application-Centric	Service- Oriented Architecture	IBM API Connect, IBM APP Connect, SAP Cloud connector	SOAP, REST, ODATA, JDBC, HTTP, JMS, TCP/IP	
Application-Centric	Legacy integration	IBM APP Connect	ODATA, JDBC, HTTP, JMS, TCP/IP	Se ne raccomanda l'uso verso PaaS o legacy system

**si consideri la lista rappresentativa dei protocolli di maggior diffusione ma non esaustiva*

5 Linee guida di sviluppo

Le best practice di sviluppo delineate dal metodo "12 Factor App" (https://12factor.net/) offrono un quadro completo per lo sviluppo di applicazioni scalabili, manutenibili e facilmente distribuibili. Di seguito vengono descritte brevemente le dodici pratiche con esempi concreti:

- **Codebase versionata**

  Principio: Una base di codice tracciata con controllo di versione, molteplici deploy.
  Esempio: Utilizzare un software version control system (nel nostro caso gitlab) per gestire il codice dell'applicazione, assicurandosi che tutte le versioni siano tracciate e che sia facile fare il deploy nei diversi ambienti del lansdcape definito.

- **Dependencies (Dipendenze)**

  Principio: Dichiarare ed isolare le dipendenze.
  Esempio: Utilizzare strumenti come pip e requirements.txt in Python o npm in Node.js per gestire le dipendenze, evitando di dipendere da librerie installate globalmente sul sistema.

- **Configurazione**

  Principio: Conservare la configurazione nell'ambiente.
  Esempio: Utilizzare variabili d'ambiente per configurazioni come URL del database, chiavi API e altre impostazioni specifiche dell'ambiente, piuttosto che hardcodare queste informazioni nel codice.

- **Backing Services (Servizi di Supporto)**

Principio: Trattare i servizi di supporto come risorse collegate.
Esempio: Configurare il database, il sistema di caching e altri servizi come risorse che possono essere sostituite senza modificare il codice dell'applicazione. Ad esempio, utilizzare DATABASE_URL per la connessione al database.

- **Build, Release, Run (Build, Rilascio, Esecuzione)**

  Principio: Separare strettamente le fasi di build e run.
  Esempio: Utilizzare sistemi CI/CD (Continuous Integration/Continuous Deployment, come Jenkins o GitLab CI) per automatizzare il processo di build, creare rilasci versionati e poi eseguire l'applicazione.

- **Processes (Processi)**

  Principio: Eseguire l'applicazione come uno o più processi senza stato.
  Esempio: Progettare l'applicazione in modo che ogni istanza di processo sia indipendente e non mantenga stato interno persistente. Utilizzare database o sistemi di caching per lo stato condiviso.

- **Port Binding**

  Principio: Esportare servizi tramite binding delle porte.
  Esempio: Un'applicazione web in Node.js utilizza app.listen(process.env.PORT || 3000) per ascoltare sulla porta specificata dalla configurazione.

- **Concurrency stateless**

  Principio: Scalare tramite il modello di processo.
  Esempio: Utilizzare un gestore di processi come Gunicorn per Python o PM2 per Node.js per scalare l'applicazione su più processi, migliorando la capacità di gestire molteplici richieste contemporaneamente.

- **Disposability**

  Principio: Massimizzare la robustezza con avviamenti rapidi e shutdown graceful.
  Esempio: Implementare meccanismi per gestire segnali di terminazione (come SIGTERM) per consentire all'applicazione di chiudere connessioni e salvare stato prima di spegnersi.

- **NoProd/Prod Parity**

  Principio: Mantenere sviluppo, staging e produzione il più simili possibile.
  Esempio: Utilizzare container Docker per creare ambienti di sviluppo che rispecchiano fedelmente la produzione, riducendo la possibilità di bug specifici dell'ambiente.

- **Log**

  Principio: Trattare i log come flussi di eventi.

Esempio: Inviare i log a stdout/stderr e utilizzare strumenti come ELK stack (Elasticsearch, Logstash, Kibana) o servizi come Splunk per raccogliere, analizzare e visualizzare i log.

- **Admin Processes**

  Principio: Eseguire i processi amministrativi come processi una tantum.
  Esempio: Eseguire migrazioni del database, backup o altre operazioni amministrative come comandi separati, ad esempio utilizzando flask db upgrade per aggiornare il database in un'applicazione Flask.

L'adozione delle pratiche del 12 Factor App consente alle organizzazioni di sviluppare e mantenere applicazioni moderne che sono resilienti, facilmente scalabili e che supportano una cultura DevOps, favorendo il continuous delivery e l'efficienza operativa.

5.1.1 Linguaggi di Programmazione

In Autostrade si indica come linguaggio di riferimento Java per lo sviluppo delle applicazioni. Tuttavia, grazie alla containerizzazione, non è un vincolo stretto e possono essere utilizzati altri linguaggi in base alle esigenze specifiche del progetto. La containerizzazione consente di standardizzare l'ambiente di runtime, indipendentemente dal linguaggio utilizzato.

5.1.2 Framework

Per quanto riguarda il framework di riferimento si raccomanda Spring Boot. Si tratta di un framework Java che semplifica lo sviluppo di applicazioni *stand-alone, production-grade*. Fornisce configurazioni predefinite per creare ed eseguire microservizi, integrandosi facilmente con diverse tecnologie e strumenti.

5.1.3 Containerizzazione

Ogni codebase deve essere consegnata con un Dockerfile definito. Il Dockerfile è un file di testo che contiene tutte le istruzioni necessarie per costruire un'immagine Docker dell'applicazione. Questo include la specifica del sistema operativo di base, le dipendenze necessarie, le configurazioni e il codice sorgente.
Si tenga conto nella valutazione di questo approccio dei benefici portati dalla containerizzazione:

- portabilità: I container possono essere eseguiti su qualsiasi ambiente che supporti Docker, garantendo la consistenza tra sviluppo, testing e produzione.
- isolamento: I container isolano l'applicazione e le sue dipendenze, prevenendo conflitti con altre applicazioni.
- scalabilità: È possibile scalare rapidamente le applicazioni aggiungendo o rimuovendo container in base alla domanda.
- gestione Facilitata: Con strumenti come Kubernetes, la gestione e l'orchestrazione dei container diventano più semplici, automatizzando il deployment, il scaling e l'operatività delle applicazioni.

5.1.4 Stateless Applications

Ogni software deve essere progettato come *stateless*, ovvero senza stato interno persistente. Questo significa che tutte le informazioni necessarie per il funzionamento dell'applicazione devono essere archiviate in sistemi esterni come database o servizi di caching. I processi dell'applicazione devono essere intercambiabili e possono essere scalati orizzontalmente senza dipendere da dati locali.

Esempio:
- Utilizzare database come PostgreSQL per memorizzare dati persistenti.
- Utilizzare Redis o Memcached per memorizzare sessioni utente o dati temporanei.

5.1.5 L'approccio di orientamento ai microservizi

Si ritiene utile per introdurre l'argomento riportare una prima definizione sull'argomento di Martin Fowler (si veda l'originale qui: https://martinfowler.com/articles/microservices.html):

"Il termine "architettura di microservizi" è nato negli ultimi anni per descrivere un modo particolare di progettare applicazioni software come suite di servizi gestibili e installabili in maniera indipendente. Sebbene non esista una definizione precisa di questo stile architetturale, esistono alcune caratteristiche comuni all'organizzazione relative a capacità aziendali, deploy automatizzato, intelligenza negli endpoint e che vede un controllo decentralizzato di linguaggi e dati."

Questo approccio consente di ottenere specifici vantaggi:

- possibilità di sviluppare software nel linguaggio più indicato, rilasciare al ritmo di sviluppo richiesto (i.e. dal business) e scalare in maniera indipendente da altri moduli/servizi;
- la decomposizione di un problema complicato («dividi et impera»), maggior riuso e parallelizzazione sviluppi;
- una maggiore leggibilità, comprensione e debug del codice;
- una migliore resilienza agli incident, un errore può impattare un microservizio ma non il resto dell'ecosistema, inoltre più specializzo e più è facile definire il processo compensato (i.e. circuit breaker).

Di contro, la sua adozione, comporta dei punti di attenzione:

- un aumento della complessità della governance;
- la conseguente richiesta di una forte automazione;
- l'assenza di ricorso a logiche di transazionalità;
- necessità di logiche di orchestrazione;
- un maggiore granularità dei servizi e un conseguente coinvolgimento di un maggior numero di componenti (es: un numero maggiore di database).

Le caratteristiche di architetture a microservizi è l'uso di infrastrutture basate su container e/o che facciano uso di servizi serverless (si veda paragrafo 4.6) e che sviluppino le logiche basate sulle funzionalità richieste o su parti di esse, in servizi indipendenti che facciano uso di *data store* dedicati e che scalino distribuendo questi servizi su più nodi di infrastruttura e anche replicandoli se necessario.
Si riporta di seguito una figura che sintetizza la differenza con l'approccio tradizionale e che vede l'uso di "monoliti"

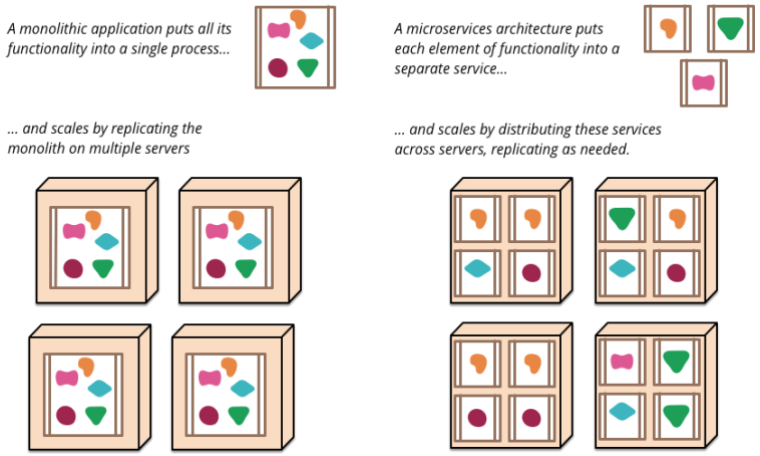


Figura 4 - Approccio a microservizi rispetto a quello "a monolite"

Per le caratteristiche appena espresse è chiaro il ricorso ad infrastrutture basate sulla containerizzazione che garantiscono portabilità, scalabilità e resilienza (es: meccanismi di "self-healing").
Volendo riassumere l'approccio questo si basa su tre caratteristiche fondamentali:

- **single purpose** : ciascun servizio dovrebbe focalizzarsi su un'unica funzionalità e farla bene
- **loose coupling** : i servizi dovrebbero conoscere il meno possibile l'uno degli altri, un cambio ad uno non dovrebbe richiedere un cambiamento sugli altri. La comunicazione tra servizi dovrebbe avvenire tramite interface pubbliche;
- **high cohesion** — ciascun servizio incorpora tutti i possibili comportamenti e i relativi dati. Se dobbiamo realizzare una nuova funzionalità tutti i cambiamenti dovrebbero essere localizzati su un solo singolo servizio.

Si riportano qui di seguito le best practice che in ASPI ci si sente di raccomandare per la progettazione di architetture di questa tipologia:

- fare uso del "The Single Responsibility Principle";
- utilizzare "data store" separati per ogni microservizio (usare approcci come "Command Query Responsibility Segregation (CQRS)");
  far uso di comunicazioni sincrone per assicurare un basso livello di accoppiamento ("loose coupling") come l'uso di interfacce asincrone o l'uso dell'approccio "event driven";
- "fail fast" e "circuit breaker pattern" per la gestione delle condizioni di fault;
- utilizzare una tecnologia di API Gateway o pub/sub per indirizzare le richieste o la gestione degli eventi;

- assicurare una “backward compatibility” per le change sulle API;
- versionare i microservizi;
- creare un team di lavoro e un processo di release specifico per ogni microservizio;
- creare efficienze organizzative (uso di standard e best practices, monitoring &logging).

Nell’implementazione di un’architettura a microservizi si segnalano alcune strategie implementative che aiutano a raggiungere le caratteristiche richieste:

- **CQRS** (Command Query Responsibility Segregation): un modello architetturale che separa la lettura e la scrittura in due modelli diversi. Ciò significa che ogni metodo dovrebbe essere un comando che esegue un'azione o una query che restituisce dati. In sintesi, un comando non può restituire dati e una query non può modificare i dati.

- **DDD (**Domain Driven Design): un approccio dello sviluppo del software che risolve problemi complessi connettendo l'implementazione ad un modello in evoluzione. Le premesse del domain-driven sono le seguenti:
  - puntare il focus primario del progetto sui domini delle entità e la loro logica.
  - basare il design sulle entità di dominio.
  - iniziare una creativa collaborazione tra tecnici ed esperti di dominio per definire in maniera iterativa un modello concettuale che possa essere applicato ai particolari problemi del caso.
  - da adottare solo nel caso di problemi complessi perché la procedura di definizione di un dominio è onerosa.

- **BC** (Bounded Context): l'obiettivo durante l'identificazione dei limiti e delle dimensioni del modello per ogni microservizio non è quello di ottenere la separazione più granulare possibile, benché sia consigliabile, se possibile, tendere a microservizi di piccole dimensioni, ma piuttosto quello di ottenere la separazione più significativa guidata dalla propria conoscenza del dominio. Nel caso di un elevato numero di dipendenze, si rende necessaria una raggruppamento per una determinata area dell'applicazione e questo determina la necessità di un singolo microservizio. Questo raggruppamento (concetto di “coesione”) è un modo per identificare come suddividere o raggruppare i microservizi.

6 AI Architecture

Questa parte ha l’obiettivo di delineare un'architettura per l'intelligenza artificiale (AI) e le sue applicazioni nel contesto aziendale. Verranno esaminate le distinzioni tra vari tipi di AI, l'approccio della "servitization" rispetto alle architetture RAG, e le diverse applicazioni dell'AI generativa (GenAI) sia per gli utenti finali che per i processi aziendali.

6.1 Distinzione tra AI e LLM

È opportuno andare a definire gli ambiti di intelligenza artificiale perché spesso e in maniera erronea si usano termini che indicano concetti diversi come sinonimi. Si riporta di seguito una figura riepilogativa e delle specifiche definizioni per evitare in futuro questo tipo di situazioni.

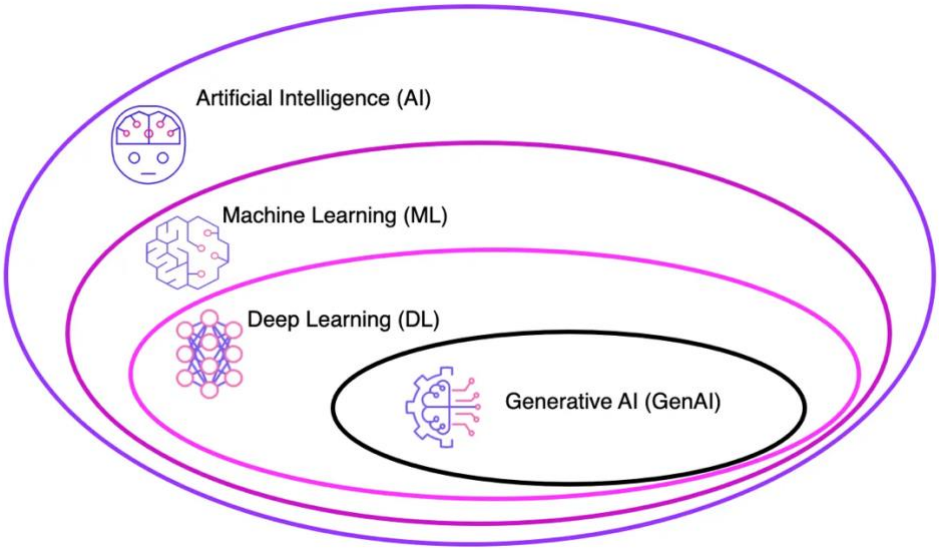


Figura 5 - AI e le sue diverse accezioni

Si parla di **AI** (Artificial Intelligence) parlando di un campo ampio che include vari sottoinsiemi e tecnologie progettate per replicare o simulare l'intelligenza umana attraverso software e sistemi. Include algoritmi di machine learning, sistemi esperti, reti neurali e altre tecniche per eseguire compiti come riconoscimento vocale, visione artificiale, e pianificazione.

Si parla di **GenAI** (Generative Artificial Intelligence) quando ci si riferisce a sistemi di intelligenza artificiale in grado di generare nuovi contenuti, idee o dati che imitano la creatività umana. Questi sistemi basano il loro funzionamento su **LLM (Large Language Model)** che è una specifica categoria di AI, basata su modelli di deep learning addestrati su grandi quantità di dati testuali. Esempi di LLM includono GPT-4. Gli LLM sono particolarmente efficaci nel comprendere e generare testo umano, supportando applicazioni come chatbot, assistenti virtuali, e traduzioni automatiche.

6.2 GenAI: tipologie ed use case

È utile riportare una prima differenziazione nell'utilizzo di GenAI:

- **GenAI per gli Utenti:** l'AI generativa per gli utenti finali si concentra sulla creazione di contenuti o risposte in base agli input degli utenti. Questo tipo di AI non necessita di una supervisione costante da parte degli sviluppatori, rendendo gli utenti autonomi nella generazione di contenuti. Un esempio sono i chatbot per assistenza clienti, strumenti di scrittura assistita, generazione automatica di immagini o musica.

- **GenAI per Processi Aziendali: l**'AI generativa applicata ai processi aziendali segue una logica di evoluzione dell'automazione dei processi robotici (RPA). Qui, l'AI genera contenuti o decisioni che ottimizzano e migliorano i flussi di lavoro esistenti. Un esempio è lo sviluppo di sistemi per l'automazione della creazione di report finanziari, la generazione di bozze di contratti, il supporto alla progettazione tecnica.

In questo senso si ritiene utile riprendere da un documento di Gartner ("When Generative AI Is and Is Not effective", 2024) quelli che sono gli *use case* in cui è consigliato adottare la GenAI:

Tipologia use case	Utilità degli attuali modelli di GenAI	Use case
Previsioni e stime	Bassa	Previsione del rischio, previsione del tasso di abbandono dei clienti, previsione vendite o dell'offerta di un mercato
Decision Intelligence	Bassa	Supporto alle decisioni,
Segmentazione/Classificazione	Media	Clustering, segmentazione clienti,classificazione oggetti
Sistemi di raccomandazione	Media	Raccomandazioni, avvisi personalizzati, azioni raccomandate
Generazione di contenuto	Alta	Generazione di testo, video e immagini, creazioni di sintesi
Interfacce conversazionali	Alta	Assistenti virtuali, chatbot, digital worker

6.3 Approccio di "Servitization" vs. Architettura RAG

Rispetto al modello operativo di gestione dell'AI è utile differenziare in due approcci:

- **Servitization (a servizi)**: la servitization trasforma il prodotto AI in un servizio continuo e migliorabile, con un focus sulla qualità del servizio e sull'innovazione continua. I vantaggi sono la scalabilità, gli aggiornamenti continui, la manutenzione facilitata, il maggiore allineamento con le esigenze aziendali e il focus sull'esperienza del cliente, dall'altra come svantaggio c'è la peculiarità di use case non coperti da un approccio standard o nella complessità nel procedere all'integrazione di dati specifici.

- **RAG (Retrieve-Augment-Generate) Architecture:** le architetture RAG combinano il recupero di informazioni, *l'augmented generation* e la generazione di contenuti. Questo approccio presenta il vantaggio di consentire una maggiore rapidità di implementazione prevdendo un accesso a informazioni che potrebbero essere più aggiornate dei dati utilizzati per addestrare il LLM (i dati contenuti nel repository di informazioni RAG possono essere continuamente aggiornati senza incorrere in costi significativi). Come svantaggi di questo apporccio sono da ricercarsi in una maggiore complessità nella gestione (ad es. la gestione della scalabilità della soluzione), nella dipendenza da interventi manuali che richiedono una manutenzione più onerosa e di un rischio di incoerenza nei risultati.

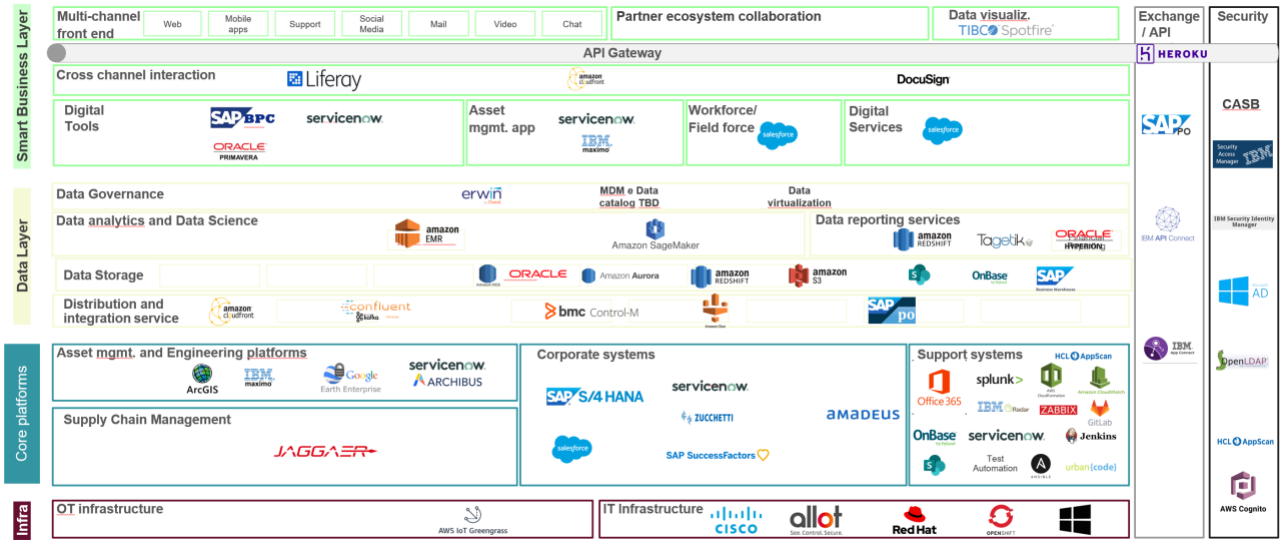
Tenuto conto degli approcci descritti, ASPI raccomanda la scelta del pattern di servitization per evitare complessità realizzative che non si traducano in valore portato al business da parte della soluzione. Per l'implementazione un approccio a servizi, consente di scomporre l'architettura AI in microservizi modulari che consentono una manutenzione, una resilienza ai guasti e un'evoluzione più agili.
Si raccomanda poi che il ciclo di vita di archietture AI sia supportato da pratiche DevOps per garantire che i modelli AI e i servizi siano sempre aggiornati e ottimizzati consentendo rilasci rapidi, feedback continuo, miglioramento della qualità del codice.
Nell'ambito dei modelli di machine learning e della loro gestione del ciclo di vita (ML ops) si raccomanda di implementare un framework di ML Ops per la gestione e la monitorizzazione dei modelli AI durante il loro ciclo di vita. Questo consente di garantire tracciabilità, monitoraggio delle performance e la capacità di risposta a fronte di degradazioni del modello.

7 Software di base supportati e piattaforme applicative

Si riporta in questo paragrafo l'elenco delle piattaforme supportate, per alcune verrà anche indicato il termine del supporto e il decomissioning in linea con il piano di trasformazione applicativa 2021-2025 che tende al target state architecture già spiegata al paragrafo 2. La lista che segue verrà aggiornata regolarmente ma all'atto della lettura potrebbe non essere esaustiva e quindi, laddove un software non sia presente, è raccomandato verificare con la struttura di CTO/IT Entreprise Architecture (mail to: enterprise.architecture@autostrade.it) la finalità dell'utilizzo per andare a valutare la sua introduzione rispetto a:

- tecnologie già presenti che soddisfino completamente o in parte le funzioni richieste;
- sostenibilità della scelta in un contesto di razionalizzazione delle tecnologie e di riduzione dei costi operativi;
- strategicità e riuso della tecnologia proposta in sostituzione o affiancamento di altre già presenti;
- richiesta di competenze specifiche di esercizio per la gestione del ciclo di vita di tale tecnologia (installazione, processo di *patching*, *change management*, gestione vulnerabilità)

Qui di seguito uno schema di sintesi del posizionamento delle varie tecnologie dettagliate e ordinate per tipologia di seguito



7.1 Smart business layer e core platforms

Vendor	Nome	Dep. model	Versione	Stato	Fine supporto	Note/Use case
LIFERAY	DXP	Onprem / IaaS	7	SUPPORTATO		Web CMS
AWS	Cloudfront	SaaS	n.a.	SUPPORTATO		CDN
DOCUSIGN	Docusign	SaaS	n.a.	SUPPORTATO		Electronic signature
ORACLE	Primavera	SaaS	n.a.	SUPPORTATO		PPM
SAP	BPC	Onprem	10.1	SUPPORTATO		Business Planning and Consolidation
SERVICE NOW	GRC (ex IRM)	PaaS	n.a.	SUPPORTATO		Modulo Service Now
SERVICE NOW	Service Now CSM	PaaS	n.a.	SUPPORTATO		Modulo Service Now

SERVICE NOW	Service Now HR	PaaS	n.a.	SUPPORTATO		Modulo Service Now
SERVICE NOW	Service Now ITBM	PaaS	n.a.	SUPPORTATO		Modulo Service Now
SERVICE NOW	Service Now ITSM	PaaS	n.a.	SUPPORTATO		Modulo Service Now
IBM	IBM Maximo	Onprem	7.6.1.3	SUPPORTATO	Prevista migrazione ad 8 nel 2025	Asset manager
IBM	IBM MAM	Onprem/ IaaS	8.6	IN DISMISSIONE		Tool di gestione sensoristica e algoritmi opere civili
SALESFORCE	Salesforce Customer Service	PaaS	n.a.	SUPPORTATO		Modulo Salesforce
SALESFORCE	Salesforce Community	PaaS	n.a.	SUPPORTATO		Modulo Salesforce
SALESFORCE	Salesforce Visual Remote Assistant	PaaS	n.a.	SUPPORTATO		Modulo Salesforce
SALESFORCE	Salesforce Field Service	PaaS	n.a.	SUPPORTATO		Modulo Salesforce
SALESFORCE	Salesforce Customer Service	PaaS	n.a.	SUPPORTATO		Modulo Salesforce
SALESFORCE	Salesforce Customer Service	PaaS	n.a.	SUPPORTATO		Modulo Salesforce
opensource	Wordpress	Onprem / IaaS	6	SUPPORTATO		Web CMS

7.2 Data layer

Vendor	Nome	Dep. model	Versione	Stato	Fine supporto	Note/Use case
ERWIN	Data modeler	Onprem	n.a.	SUPPORTATO		Data modeling
ZEENEA	Data catalog	SaaS	n.a.	SUPPORTATO		Data catalog
TIBCO	Spotifire	IaaS	11.x in poi	SUPPORTATO		Data visualization, standard reporting, self service reporting
TIBCO	Data virtualization	IaaS	8.x in poi	SUPPORTATO		Data virtualization
AWS	AWS EMR	PaaS	n.a	SUPPORTATO		
AWS	AWS Aurora - PostgreSQL	PaaS	n.a	SUPPORTATO		Database Management System (relazionale)
opensource	MongoDB	Onprem / IaaS	4.x in poi	SUPPORTATO		Database Management System (noSQL)
AWS	DocumentDB	PaaS	n.a			Database Management System (noSQL)
AWS	RDS PostgresQL	PaaS	n.a	SUPPORTATO		Database Management System (relazionale)
AWS	RDS Oracle	PaaS	n.a	SUPPORTATO		Database Management System (relazionale)
AWS	Redshift	PaaS	n.a.	SUPPORTATO		DWH, Data visualization, standard reporting
AWS	Opensearch	SaaS	1.1	SUPPORTATO		Gestione dati non strutturati
HYLAND	Onbase Cloud	PaaS	22.1	SUPPORTATO		Document Management

ESRI	Arcgis	Onprem	11.1.0	SUPPORTATO		GIS Management System
JAGGAER	Jaggaer One	SaaS	24.1	SUPPORTATO		Processi di procurement
SAP	SAP Success Factor	SaaS	n.a	SUPPORTATO		Processi organizzazione e personale
SAP	SAP ECC	Onprem	6.05	IN MIGRAZIONE	31/12/2022	SAP Core services
SAP	SAP Portal	Onprem	7.0.1	IN MIGRAZIONE	31/12/2022	Software di portale
SAP	SAP BW	Onprem	7.3	IN DISMISSIONE (a favore SAP DATASPHERE)	31/12/2022	Software di DWH
SAP	SAP SOLMAN	Onprem	7.2	IN MIGRAZIONE		
ORACLE	Database	Onprem	11g-12c-18c	PARZIALMENTE SUPPORTATO	31/12/2023	
AWS	RDS MySQL	PaaS	n.a	PARZIALMENTE SUPPORTATO		Supportato solo per wordpress
SAP	Business Object	Onprem	9.0	IN DISMISSIONE	31/12/2023	
SAP	SAP Analytics Cloud	PaaS		SUPPORTATO		Data visualization
SAP	SAP Datasphere	PaaS		SUPPORTATO		DWH
IBM	DB2 Warehouse	Onprem	11.1	IN DISMISSIONE	31/12/2023	DWH, Data visualization, standard reporting
HYLAND	On base	Onprem	19.8	IN MIGRAZIONE	31/12/2021	Document Management
HYLAND	Alfresco ECM	Onprem	6.2	IN DISMISSIONE	31/12/2025	Document Management
IBM	DB2 LUW	Onprem	11.1	IN DISMISSIONE	31/12/2025	Database Management System (relazionale)
MICROSOFT	SQL Server Standard Edition	Onprem	2008 R2	PARZIALMENTE SUPPORTATO	31/12/2022	Database Management System (relazionale)
MICROSOFT	SQL Server Enterprise Edition	Onprem	2017	PARZIALMENTE SUPPORTATO	31/12/2022	Database Management System (relazionale)
MICROSOFT	Azure SQL	PaaS	N.A.	SUPPORTATO		Database Management System (relazionale)
MICROSOFT	Azure PostgreSQL	PaaS	N.A.	SUPPORTATO		Database Management System (relazionale)

7.3 Software di integrazione

Vendor	Nome	Dep. model	Versione	Stato	Fine supporto	Note/Use case
SALESFORCE	Heroku	PaaS	n.a.	IN DISMISSIONE	31/12/2025	Solo per funzione offloading CRM
SAP	PI	Onprem	7.11	IN DISMISSIONE		API management
SAP	Cloud Connector			SUPPORTATO		Integration software
IBM	API Connect	Onprem/IaaS	2018.4.1	SUPPORTATO	30/04/2023	API management
IBM	APP Connect	Onprem/IaaS	11.0.0.9	SUPPORTATO		API management
CONFLUENT	Kafka	Onprem	7.0.1	IN DISMISSIONE (parte connettori)	31/12/2024	Software pub/sub
CONFLUENT	Kafka cloud	SaaS	n.a.	SUPPORTATO		Software pub/sub as a service
IBM	Sterling Integrator	Onprem	6.1	SUPPORTATO		File transfer
AWS	AWS Glue	SaaS	n.a.	SUPPORTATO		Data integration, ETL
IBM	Data Replication	Onprem	14.x in poi	SUPPORTATO		Change data-capture
IBM	Datastage	Onprem	11.7	IN DISMISSIONE	31/12/2021	ETL
AWS	DMS	SaaS	n.a.	SUPPORTATO		Tool CDC

7.4 Software DevSecOps

Vendor	Nome	Dep. model	Versione	Stato	Fine supporto	Note/Use case
Gitlab	Gitlab	Onprem	11.10.4	SUPPORTATO		Source code repository
	Jenkins	Onprem	2.289.3	SUPPORTATO		Orchestration
SONATYPE	Nexus	Onprem	3.36.0	SUPPORTATO		Repository Artifactory
SONARSOURCE	Sonarqube	IaaS	8.9.1	SUPPORTATO		Quality Code
HCL	AppScan	Onprem	10.0	SUPPORTATO		SAST,DAST
	ArgoCD	PaaS	2.1.6	SUPPORTATO		Deployment Manager per Kubernetes
	Harbor	PaaS	2.4.0	SUPPORTATO		Container Registry
IBM	Urbancode	Onprem	7.1	SUPPORTATO		Deployment Manager

7.5 Software di Infrastruttura e di supporto

Vendor	Nome	Dep. model	Versione	Stato	Fine supporto	Note/Use case
AWS	S3	SaaS	n.a.	SUPPORTATO		Storage service
AWS	EC2	IaaS	n.a.	SUPPORTATO		Compute service
VMWARE	VMWARE	Onprem	xx	SUPPORTATO		Servizio di virtualizzazione
ZABBIX	Zabbix	Onprem	5	SUPPORTATO		Monitoraggio tecnico
AWS	AWS Cloudwtach	SaaS	n.a.	SUPPORTATO		Monitoraggio tecnico
DYNATRACE	Dynatrace	SaaS	n.a.	SUPPORTATO		Monitoraggio tecnico, funzionale e real user monitoring
REDHAT	Redhat Enterpise Linux			SUPPORTATO		OS
REDHAT	RAHCM	xx		SUPPORTATO		
REDHAT	Openshift	Onprem	4.8	SUPPORTATO		Software gestione container onprem, k8s
AWS	AWS ROSA	PaaS	n.a.	IN DISMISSIONE		Software gestione container, k8s, as a service
AWS	AWS Corretto	IaaS	8	SUPPORTATO		Java virtual machine
ORACLE	ORACLE JVM	Onprem	8	SUPPORTATO		Java virtual machine
ORACLE	ORACLE Linux	Onprem	8	SUPPORTATO		OS
opensource	Apache webserver	Onprem/IaaS	2.x	SUPPORTATO		Web server
opensource	Apache tomcat	Onprem/IaaS	1.8	SUPPORTATO		Application server java
BMC	Control-M	Onprem	9	SUPPORTATO		Software di schedulazione
AWS	AWS Batch	PaaS	n.a	SUPPORTATO		Batch as a service
MICROSOFT	Microsoft Windows Server	Onprem	2016	IN DISMISSIONE	31/12/2022	OS
MICROSOFT	Microsoft IIS	Onprem	10	IN DISMISSIONE	31/12/2022	Web server, application server
IBM	AIX	Onprem	7	IN DISMISSIONE	31/12/2022	OS

IBM	Websphere	Onprem	8,9	IN DISMISSIONE	31/12/2021	Application Server java
AWS	DMS	SaaS	n.a.	SUPPORTATO		Database Migration
AWS	EKS	PaaS	n.a.	SUPPORTATO		Software gestione container, k8s.
AWS	Storage Gateway	PaaS	n.a.	SUPPORTATO		Share file/object server

7.6 Software di IT security

Vendor	Nome	Dep. model	Versione	Stato	Fine supporto	Note/Use case
IBM	ISAM	Onprem		SUPPORTATO	mar-25	SSO
IBM	IGI	Onprem		In Dismissione	mar-25	Authorization Platform
Oneidentity	Oneidentity Safeguard	Onprem		SUPPORTATO		PAM
MICROSOFT	Active Directory	Onprem		SUPPORTATO		LDAP server
AWS	Cognito	SaaS		SUPPORTATO		Security proxy
OPENLDAP PROJECT	OpenLDAP	Onprem		SUPPORTATO		LDAP server
IBM HCL	AppScan	SaaS		SUPPORTATO		Code Analysis
Entrust	Cetificate Autority	SaaS		SUPPORTATO	nov-24	Rilascio Certificati
MICROSOFT	Active Directory Cetificate Services	Onprem		SUPPORTATO		Rilascio Certificati
Hashicorp	Vault	Onprem		SUPPORTATO	nov-26	Rilascio Certificati
Hashicorp	Terraform	SaaS		SUPPORTATO	nov-26	Rilascio Certificati
IBM	Qradar	Onprem		Sostituzione		SIEM
IBM	Resilient	Onprem		In Dismissione		SOAR
Palo Alto	Cortex	SaaS		SUPPORTATO	mag-27	SOAR
Crowdstrike	Cloud Security	SaaS		SUPPORTATO		Cloud Protection
Crowdstrike	Endpoint Security	Endpoint		SUPPORTATO		Endpoint Protection
Gitlab	Gitlab	Onprem		SUPPORTATO		Code Repository
Lookout	Mobile Security	SaaS		SUPPORTATO		Mobile protection
Proofpoint	Proofpoint Security Awareness Training (PSAP)	SaaS		SUPPORTATO		Cyber Awerness
Proofpoint	Mailgateway	SaaS		SUPPORTATO		Sistema di Antispam
Tenable	Tenable.SC	Ibrido		SUPPORTATO		Vulnerability Management
Picus	BAS	SaaS		SUPPORTATO		Assesment policy, breach attack simulation
Pentera	SVP	SaaS		SUPPORTATO		Security Validation (Penetration Test)
Netskope	Netskope	SaaS		SUPPORTATO		Proxy utente

Teampass	Teampass	Onprem		SUPPORTATO		b
MISP	TIP	Onprem		SUPPORTATO		Threat intelligence
Bitsight	Bitsight	SaaS		SUPPORTATO		Threat intelligence
CoreView	CoreView	SaaS		SUPPORTATO		Licensing Management
SNOW	Snow Atlas	SaaS		SUPPORTATO		Software Inventory
IriusRisk	Threat Modeler	SaaS	n/a	SUPPORTATO	31/06/2027	Threat Modeler
AWS	AWS WAF	SaaS	n/a	SUPPORTATO		WAF
AWS	Cloud Shield	SaaS	n/a	SUPPORTATO		Anti-DDoS
Fortinet	FortiClient VPN	Onprem		SUPPORTATO		Servizio VPN
Wiki.JS	Wiki.JS	Onprem	n/a	SUPPORTATO		Wiki Security
Varonis	DataAlert	Onprem		SUPPORTATO	dic-24	Data classification